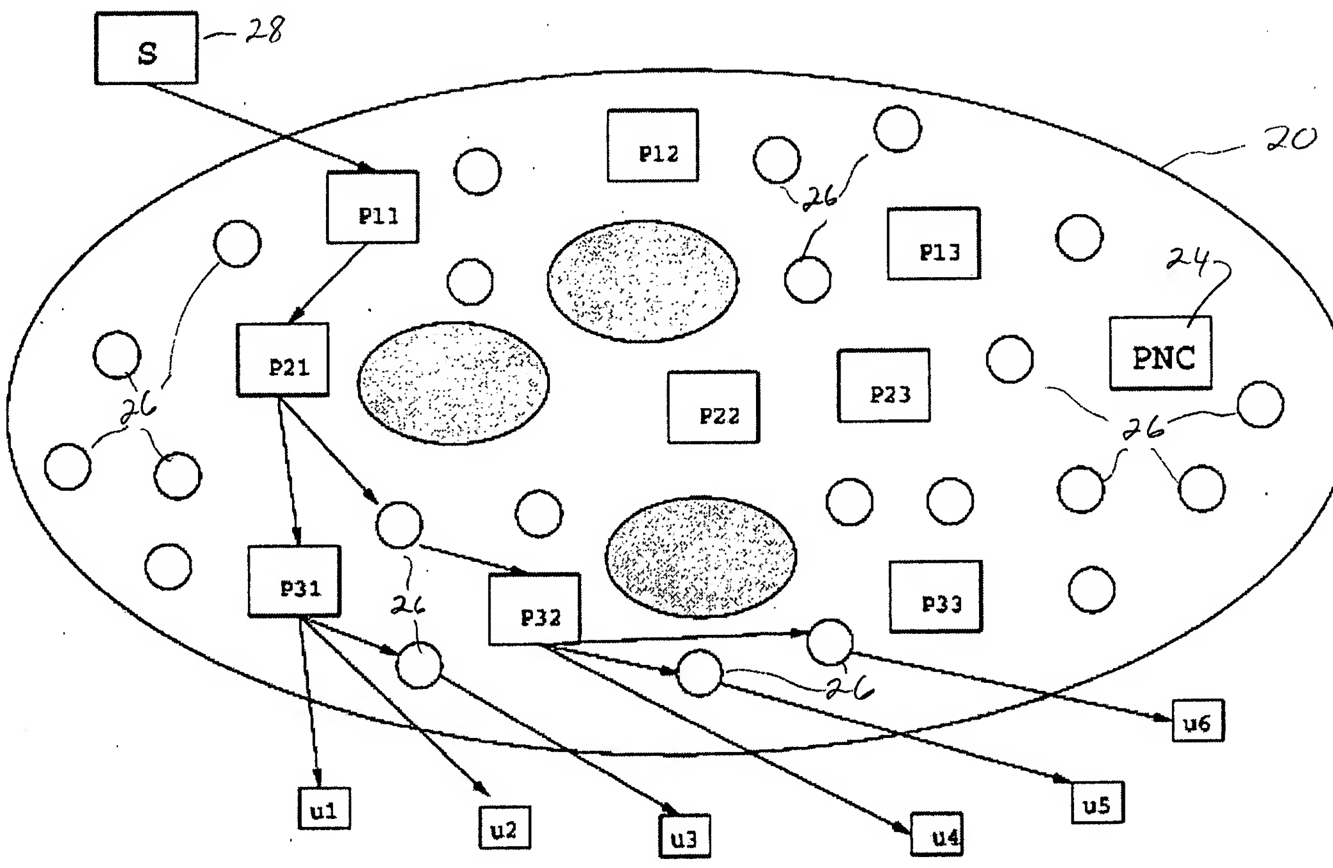
S
28
20
P11
P12
P13
P21
P22
P23
P31
P32
P33
PNC
24
26
u1
u2
u3
u4
u5
u6

FIG 1(a)

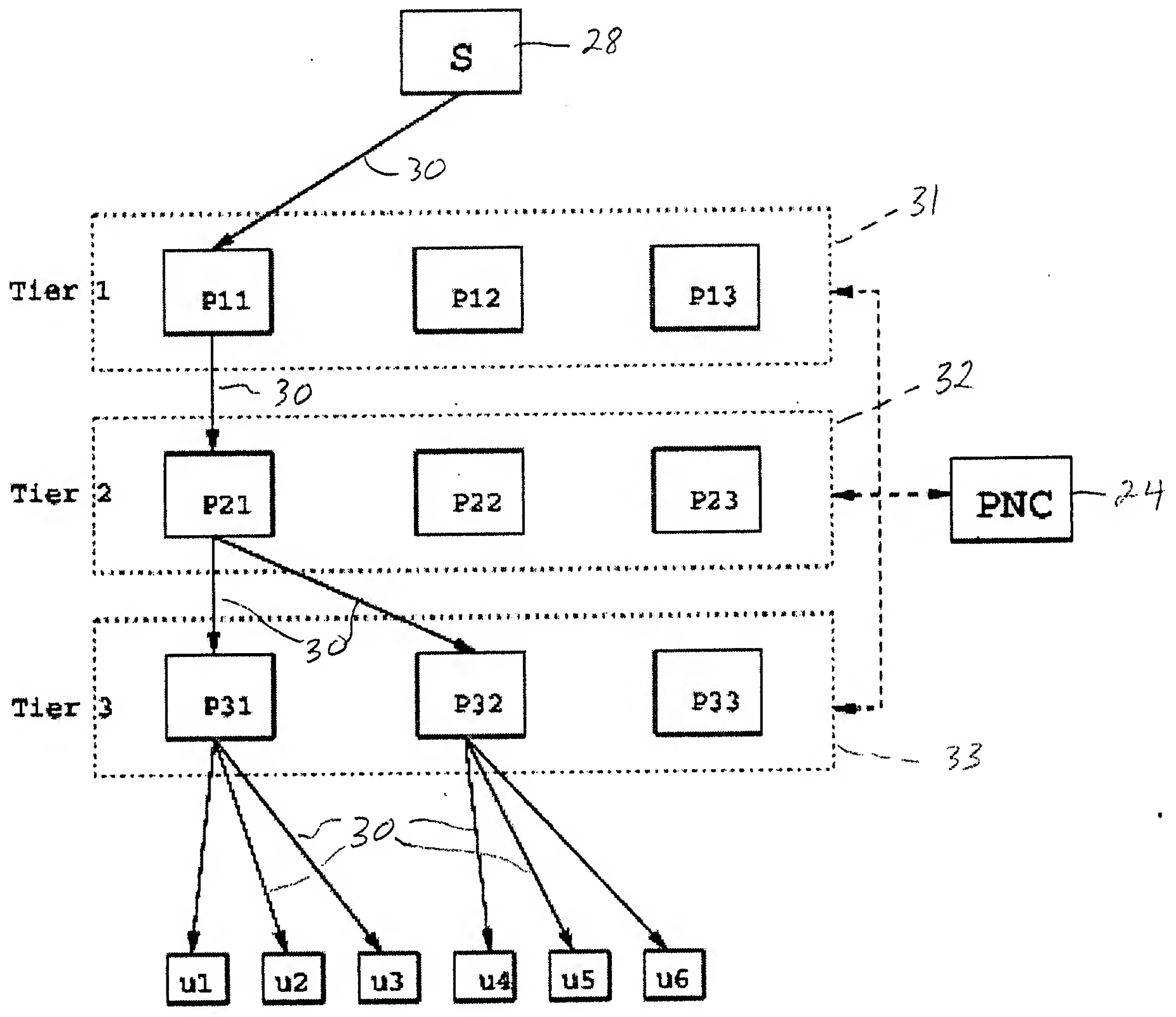
S
28
30
31
32
33
24
PNC
Tier 1
Tier 2
Tier 3
P11
P12
P13
P21
P22
P23
P31
P32
P33
u1
u2
u3
u4
u5
u6

FIG 1(b)

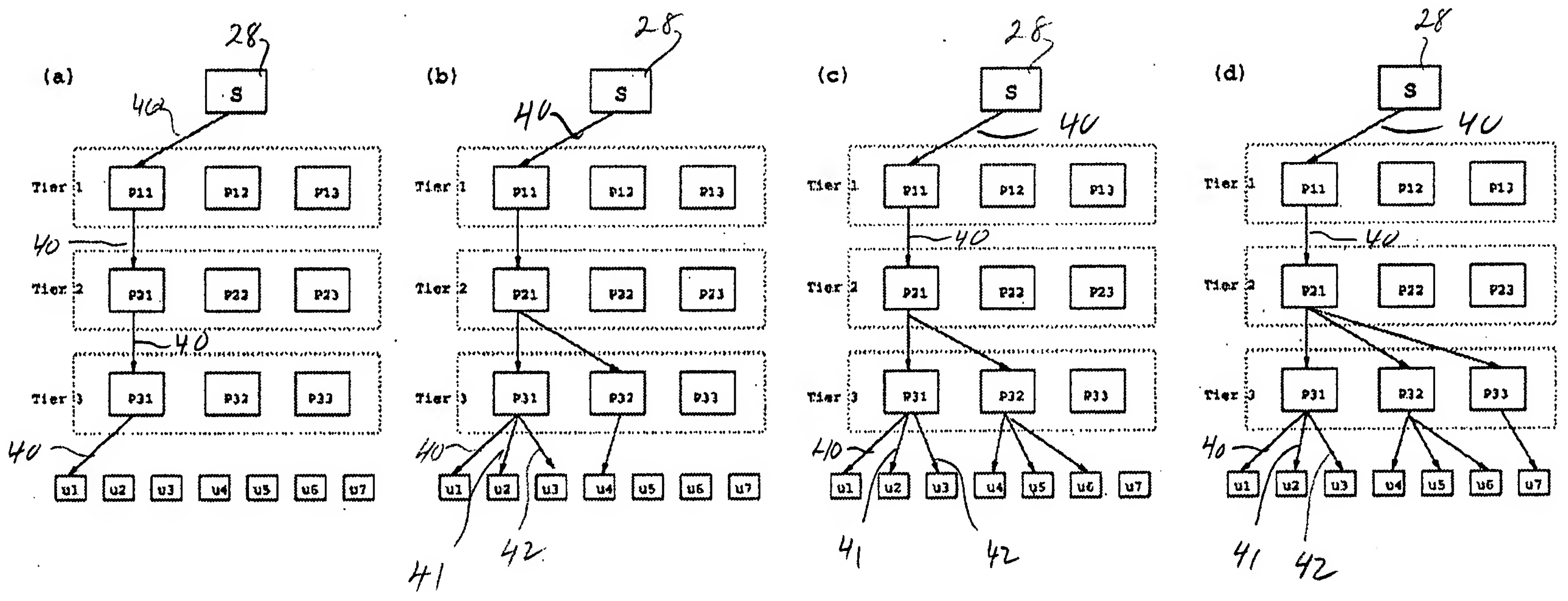
(a)
(b)
(c)
(d)
S
28
40
41
42
Tier 1
Tier 2
Tier 3
P11
P12
P13
P21
P22
P23
P31
P32
P33
u1
u2
u3
u4
u5
u6
u7

FIG 2

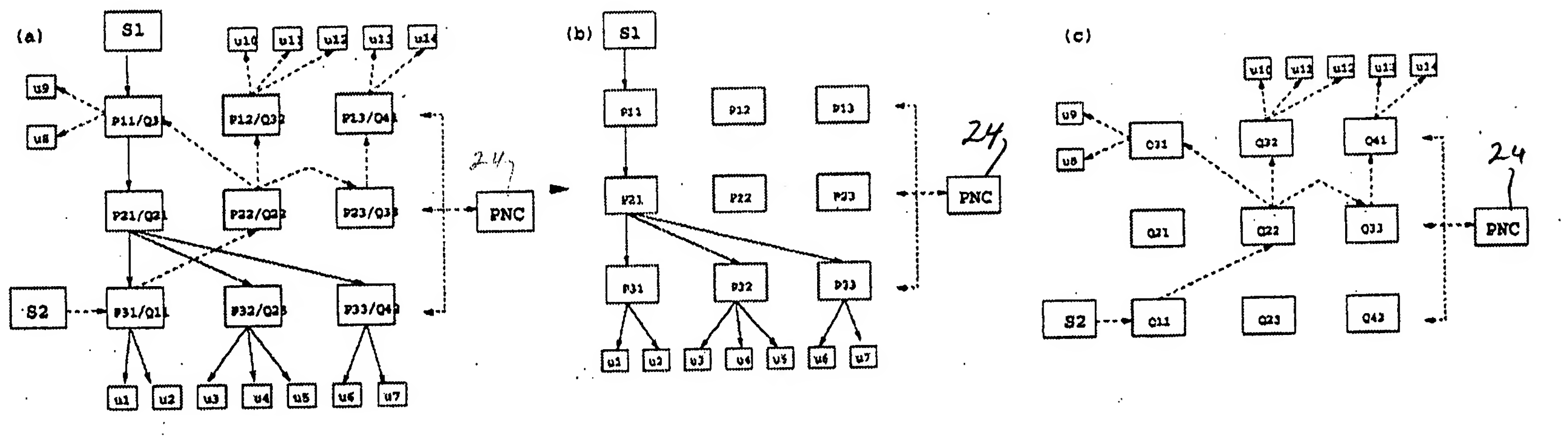
(a)
S1
S2
u9
u8
P11/Q31
P12/Q32
P13/Q41
P21/Q21
P22/Q22
P23/Q33
P31/Q11
P32/Q23
P33/Q42
u10
u11
u12
u13
u14
u1
u2
u3
u4
u5
u6
u7
PNC
24
(b)
S1
P11
P12
P13
P21
P22
P23
P31
P32
P33
u1
u2
u3
u4
u5
u6
u7
PNC
24
(c)
S2
u9
u8
Q31
Q32
Q41
Q21
Q22
Q33
Q11
Q23
Q42
u10
u11
u12
u13
u14
PNC
24

FIG 3

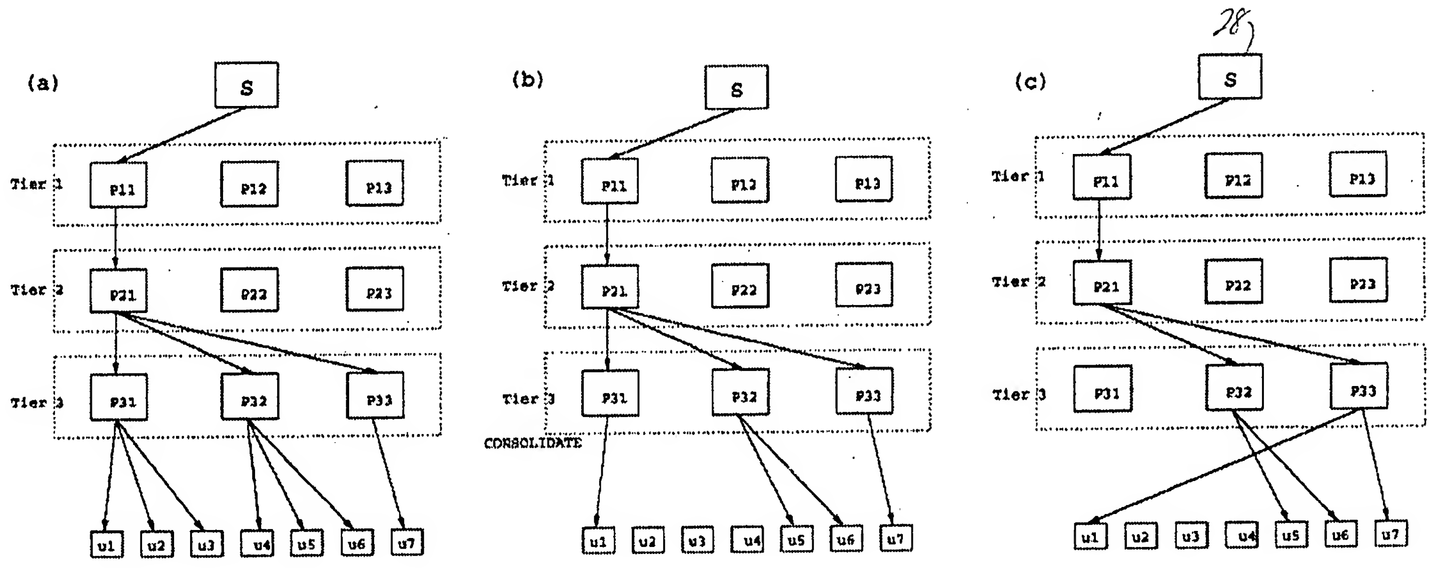
(a)
S
Tier 1
P11
P12
P13
Tier 2
P21
P22
P23
Tier 3
P31
P32
P33
u1
u2
u3
u4
u5
u6
u7
(b)
S
Tier 1
P11
P12
P13
Tier 2
P21
P22
P23
Tier 3
P31
P32
P33
CONSOLIDATE
u1
u2
u3
u4
u5
u6
u7
(c)
28
S
Tier 1
P11
P12
P13
Tier 2
P21
P22
P23
Tier 3
P31
P32
P33
u1
u2
u3
u4
u5
u6
u7

Fig 4

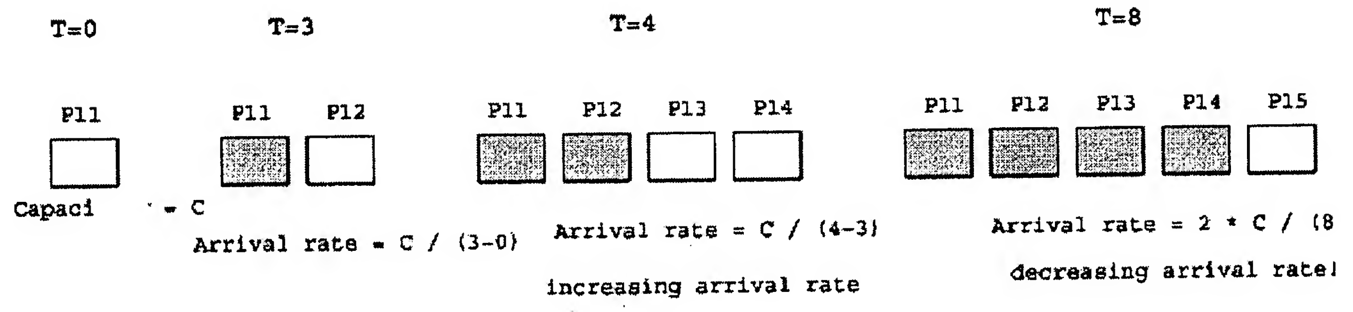
T=0
T=3
T=4
T=8
P11
P11
P12
P11
P12
P13
P14
P11
P12
P13
P14
P15
Capaci · = C
Arrival rate = C / (3-0)
Arrival rate = C / (4-3)
increasing arrival rate
Arrival rate = 2 * C / (8
decreasing arrival rate!

FIG 5

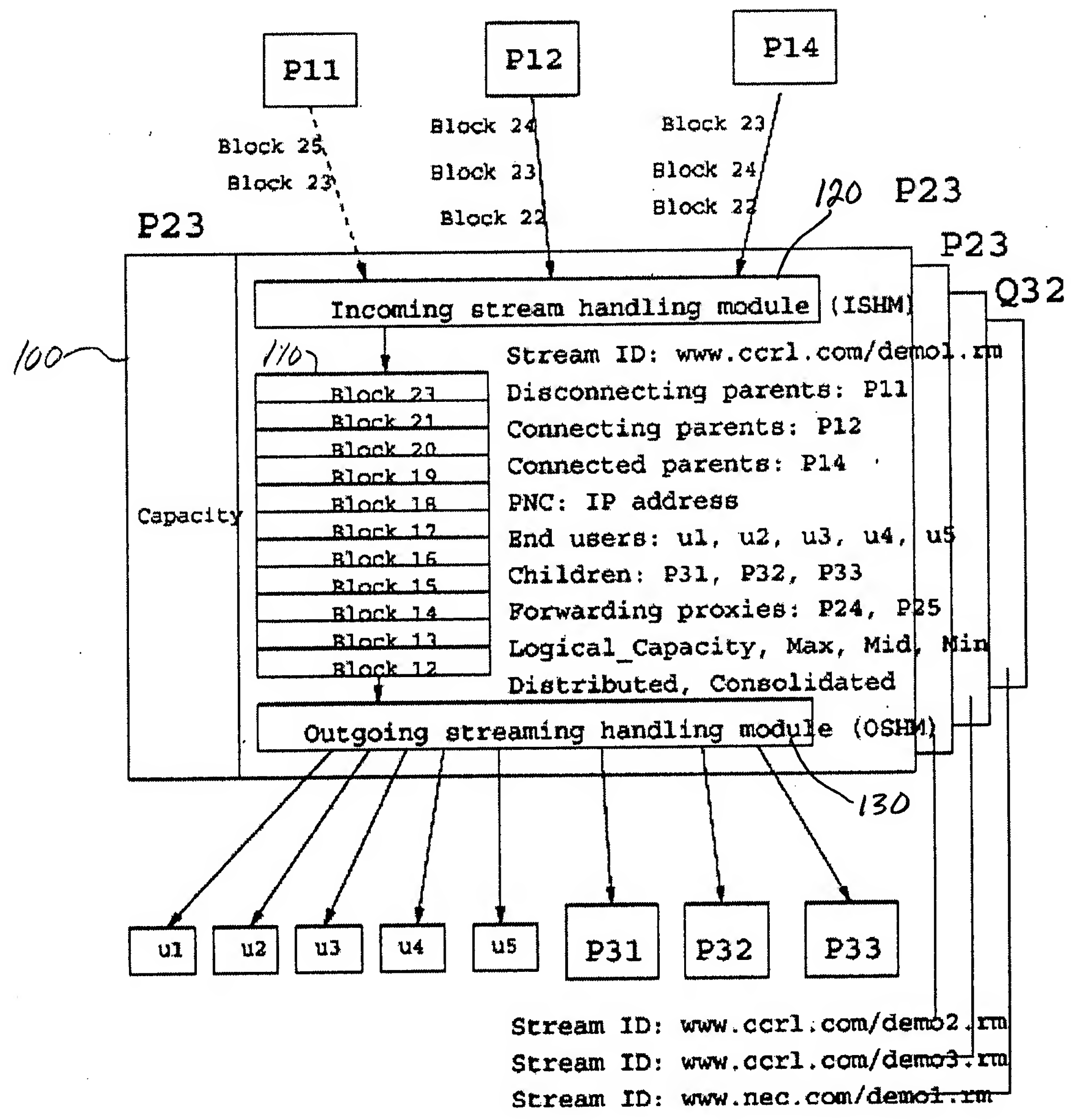
P11
P12
P14
Block 25
Block 23
Block 24
Block 23
Block 22
Block 23
Block 24
Block 22
120
P23
P23
P23
Q32
100
Capacity
Incoming stream handling module (ISHM)
110
Block 23
Block 21
Block 20
Block 19
Block 18
Block 17
Block 16
Block 15
Block 14
Block 13
Block 12
Stream ID: www.ccrl.com/demo1.rm
Disconnecting parents: P11
Connecting parents: P12
Connected parents: P14
PNC: IP address
End users: u1, u2, u3, u4, u5
Children: P31, P32, P33
Forwarding proxies: P24, P25
Logical_Capacity, Max, Mid, Min
Distributed, Consolidated
Outgoing streaming handling module (OSHM)
130
u1
u2
u3
u4
u5
P31
P32
P33
Stream ID: www.ccrl.com/demo2.rm
Stream ID: www.ccrl.com/demo3.rm
Stream ID: www.nec.com/demo1.rm

FIG 6

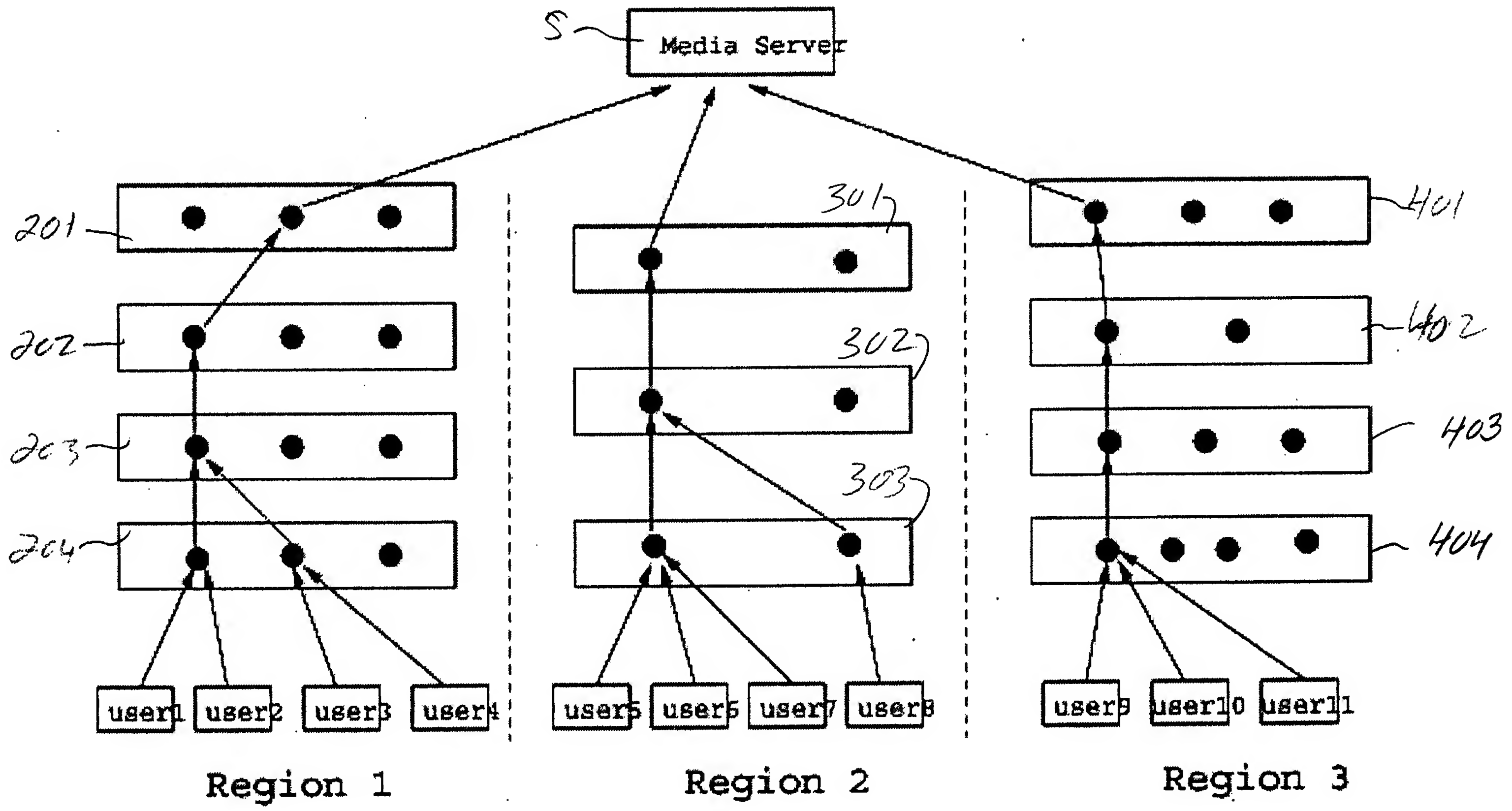
5
Media Server
201
202
203
204
301
302
303
401
402
403
404
user1
user2
user3
user4
user5
user6
user7
user8
user9
user10
user11
Region 1
Region 2
Region 3

FIG 7

```
MODULE DynamicMultiSourceProxyServer;
/* Data Structure to maintain connection states */
struct ConnectionState{
    int                 State; /* IDLE, CONNECTING */
    URL                 StreamSource; /* Source Identifier */
    NetHost             Parent; /* Parent Node of this Connection */
    int                 ChildCount; /* Number of children */
    LIST OF NetHost     Children; /* Children of this Connection */
    LIST OF NetHost     Waiting; /* Awaiting parent connection */
    int                 Load, Dist, Cons, Max, LinRate, LoffRate;
                            /* Connection Management; */
} struct ProxyParent{
    URL                 StreamSource; /* Source Identifier */
    NetHost             ParentCache; /* my parent for this source? */
}
SET OF ConnectionState  Conns; /* global variables */
            /* INITIALIZATION HERE */
/* Event Handling */
MONITOR(P) from sender S for sourceURL;
LOGIN(params) from sender S for sourceURL: See Figure 9
LOGOFF(params) from sender S for sourceURL: See Figure 10
CONNECTED() from S for sourceURL;
ConnRefused() from S for sourceURL;
SwitchToParent(P) from PNC for sourceURL;
```

Figure 8

```
LOGIN(params) from sender S for sourceURL:
{
 Let CC be in Conns such that CC.StreamSource = sourceURL;
 if there is no ConnectionState then
        · setup the structure to maintain the ConnectionState;
        · update load information appropriately;
 endif
 update CC.Load, CC.LoginRate using double smoothing;
 if (New ConnectionState)
        send(LOGIN(params+MyParams)) to CC.Parent for sourceURL;
        mark this request as WAITING for
        a connection setup with the parent Proxy;
 else if (ConnectionState indicates login requests to parent is pending)
        mark this request as WAITING for
        a connection setup with the parent Proxy;
 else /* parent connection already exists */
        SetUpLocalConnection(params, S); /* allocate buffers etc. */
        send(CONNECTED) to S for sourceURL; }
 update CC.Load, CC.LoginRate using double smoothing;
 if (OVERFLOW possible)
 /* check if the current load+login rate-logoff rate will cause overflow */
        send(DISTRIBUTE(MyParams)) to ProxyNetworkCoordinator;
}
```

Figure 9

```
LOGOFF(params) from sender S for sourceURL:
{
 Find entry CC in Conns s.t. Entry.StreamSource=sourceURL;
 TearDownLocalConnection(params, S); /* deallocate buffers etc. */
 Remove S from the list of children of CC;
 Conn.Children=Conn.Children \ {S};
 update load parameters: Load, LogoffRate using double smoothing;
 if UNDERFLOW possible
 /* check if the current load+login rate-logoff rate will cause overflow */
      send(CONSOLIDATE(MyParams)) to ProxyNetworkCoordinator
      for sourceURL;
 endifif (this is the last user to logoff)
      send(LOGOFF(params,MyParams)) to Conn.Parent for sourceURL;
      DEALLOCATE Conn;
 endif }
```

Figure 10

```
MODULE DynamicMultiSourcePNC;
struct Proxy{
     NetHost                  Id;
     int                      State; /* FREE, INUSE, FAILED */
     int                      Tier; /* Tier Identifier */
     NetHost                  Parent;
     List of NetHost          Children;
}
struct SourceProxyPair{
     NetHost                  StreamSource;
     Proxy                    Overlay[NTiers][NProxies];
     INFO                     ProxyMaint[NTiers];
} /* The proxies are maintained in a layered manner */
SET OF SourceProxyPair ProxyNet; /* Global Variables */
 BEGIN
/* Initialization */
/* Initiate Link Monitoring Activity */
/* Event Handling */
DISTRIBUTE() from S for sourceURL: See Figure 12.
CONSOLIDATE() from S for sourceURL: See Figure 14.
END.
```

Figure 11

```
DISTRIBUTE() from S for sourceURL:
sp ⇐ stability period
$SysLinRate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LinRate_p$
$SysLoffRate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LoffRate_p$
if $((SysLinRate - SysLoffRate) \cdot sp) \geq \sum_{p \in Proxies[S.tier]}(Max_p - Load_p)$
    $Load_1 = ((SysLinRate - SysLoffRate) \cdot sp) - \sum_{p \in Proxies[S.tier]}(Max_p - Load_p)$
    $Load_2 = SysLinRate \cdot \Delta T$;
    $AnticipatedLoad = MAX(Load_1, Load_2)$;
    FIND $m$ proxies in S.tier such that $\Sigma_{i=1}^{m} Max_i \geq AnticipatedLoad$ and $m$ is minimum;
    Let this set be $\mathcal{P} = \{P_1, P_2, ..., P_m\}$;
else /* the current load can be handled by currently active proxies */
    $\mathcal{P} = CreateServerFarmFromActiveProxies()$;
    if $\mathcal{P} = \emptyset$ {
        $AnticipatedLoad = SysLinRate \cdot \Delta T$;
        FIND $m$ proxies in S.tier such that $\Sigma_{i=1}^{m} Max_i \geq AnticipatedLoad$ and $m$ is minimum;
        Let this set be $\mathcal{P} = \{P_1, P_2, ..., P_m\}$;
    }
for (each proxy $T \in \mathcal{P}$) do
{
    Activate T in p.Overlay;
    /*find parent; use a round robin allocation if multiple parents*/
    P = FindCurrentActiveProxy(p.Overlay, ParentTier(T.tier));
    /* the following steps maintains the active part of the overlay */
    p.Overlay[P.i][P.j].Children=* ∪ {T};
    p.Overlay[T.i][T.j].Parent=*∪{P};
    if (T is at the lowest level) then add T to the DNS;
    send(SwitchToParent(P)) to T for sourceURL; }
```

Figure 12

```
CreateServerFarm:
    for all active proxis $S$ do{
        find minimum $MIN$ such that
            $((\frac{SysLinRate}{MIN} - Loffrates_S) \cdot sp) \leq Max_S - Load_S$;
        ADD $\langle MIN, S \rangle$ to the HashTable;
    }
    $S = \emptyset$;
    $count = 0$;
    for $(i = 0; i < n; i++)$ do{
        Let the set of proxies with hash value i be $S'$;
        $count = count + |S'|$;
        $S = S \cup S'$;
        if $(count \geq i)$ RETURN $S$;
    }
    RETURN {};
```

Figure 13

```
CONSOLIDATE() from S for sourceURL:
    $sp \Leftarrow$ stability period
    $SysLinRate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LinRate_p$
    $SysLoffrate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LoffRate_p$
    if $\left((SysLinRate - SysLoffRate) \cdot sp \geq \sum_{p \in ProxiesinS.Tier \wedge p \neq S} (Max_p - Load_p)\right)$ S is deactivated;
```

Figure 14.